J Comb Optim (2009) 17: 323–338
DOI 10.1007/s10878-007-9114-0

Single machine batch scheduling with release times

Beat Gfeller · Leon Peeters · Birgitta Weber · Peter Widmayer

Published online: 22 November 2007

Abstract Motivated by a high-throughput logging system, we investigate the single machine scheduling problem with batching, where jobs have release times and processing times, and batches require a setup time. Our objective is to minimize the total flow time, in the online setting. For the online problem where all jobs have identical processing times, we propose a 2-competitive algorithm and we prove a corresponding lower bound. Moreover, we show that if jobs with arbitrary processing times can be processed in any order, any online algorithm has a linear competitive ratio in the worst case.

Keywords Batch scheduling · Online algorithms · Competitive analysis

A preliminary version of a part of this paper was presented at the 31st International Symposium on Mathematical Foundations of Computer Science (MFCS 2006). We gratefully acknowledge reviewers' comments that helped to improve the presentation of this work.

Supported by the Swiss SBF under contract no. C05.0047 within COST-295 (DYNAMO) of the European Union.

Research carried out while B. Weber was affiliated with the Institute of Theoretical Computer Science, ETH Zurich.

B. Gfeller (✉) · L. Peeters · B. Weber · P. Widmayer
Institute of Theoretical Computer Science, ETH Zurich, Zurich, Switzerland
e-mail: gfeller@inf.ethz.ch

L. Peeters
e-mail: peeters@inf.ethz.ch

B. Weber
e-mail: birgitta.weber@unilever.com

P. Widmayer
e-mail: widmayer@inf.ethz.ch

1 Introduction

The study in this paper is motivated by the real world problem of saving a log of actions in a high throughput environment. Many actions are to be carried out in rapid succession, and in case of a system failure the log can identify which actions have been carried out before the failure and which have not. Keeping such a log can be existential for a business, for example when logging the trading data in a stock brokerage company.

Logging takes place on disk and is carried out by a storage system that accepts write requests. When a process wants its data to be logged, it sends a log request with the log data to the storage system and waits for the acknowledgement that the writing of the log data has been completed. For the log requests that arrive over time at the storage system, there is only one decision the system is free to make: What subset of the requested, but not yet written log data should be written to disk in a single large write operation to make the whole system as efficient as possible? After the chosen large write operation is complete, the system instantaneously sends acknowledgements to all processes whose requests have been satisfied.

The difficulty in the above question comes from the fact that log data come in all sizes (number of bits or blocks to be stored), that writing several log data in a single shot is faster than writing each of them individually (due to the disk hardware constraints), and that a process requesting a write has to wait for the acknowledgement (of the completion of the large write operation) before it can continue. Based on an experimental evaluation of writing data to disk, we assume the writing time for a number of data blocks to be linear in that number, plus an additive constant (for the disk write setup time). Our objective is to minimize the sum over all requests of the times between the request's arrival and its acknowledgement. We ignore the details of a failure and its recovery here, and are not worrying about (the potential loss of) unsatisfied write requests.

1.1 Single machine scheduling with batching

Viewing the storage system as a machine, the log requests as jobs, and the write operations as batches, this problem falls into the realm of scheduling with batching (see the overview by Potts and Kovalyov 2000). More precisely, in the usual batch scheduling taxonomy of Potts and Kovalyov (2000) we deal with a *family scheduling problem* with batching on a single machine where all the jobs belong to the same family. The machine processes the jobs consecutively, since the log data are stored consecutively in time on the disk (as opposed to simultaneously), and each batch of jobs requires a constant (disk write) setup time. As all log requests in a single write are simultaneously acknowledged at the write completion time, the machine operates with *batch availability*, meaning that each job in the batch completes only when the full batch is completed.

In more formal terms, we model the storage system as a single machine, and the log requests as a set of jobs $J = \{1, \ldots, n\}$. The arrival times of the log requests at the storage system then correspond to job release times r_j, $j \in \{1, \ldots, n\}$. Further, each job $j \in J$ has a processing time p_j on the machine, representing the block size

of the log request. The grouping of the log requests into write operations is modeled by the batching of the jobs into a partition $\sigma = \{\sigma_1, \ldots, \sigma_k\}$ of the jobs $\{1, \ldots, n\}$, where σ_u represents the jobs in the u-th batch, k is the total number of batches, and we refer to $|\sigma_u|$ as the *size* of batch u, defined as the number of jobs in σ_u. Unless stated otherwise, we assume that the batch size is not limited. We denote the starting time of batch σ_u by T_u, with $r_j \leq T_u$ for all $j \in \sigma_u$. Starting at T_u, the batch requires a constant setup time s for preparing the disk write, and further a total processing time $\sum_{j\in\sigma_u} p_j$, for writing the logs on the disk. Thus, each batch σ_u requires a total *batch processing time* $P_u = s + \sum_{j\in\sigma_u} p_j$. The consecutive execution of batches on the single machine translates into $T_u + P_u \leq T_{u+1}$ for $u = 1, \ldots, k-1$. Because of batch availability, each job $j \in \sigma_u$ completes at time $C_j = T_u + P_u$, and takes a flow time $F_j = C_j - r_j$ to be processed. This job flow time basically consists of two components: first a *waiting time* $T_u - r_j \geq 0$ that the job waits before batch σ_u starts, followed by the batch processing time P_u. Finally, as mentioned above, our objective is to minimize the total job flow time $\mathcal{F} = \sum_{j=1}^{n} F_j$.

We refer to this scheduling problem as the BATCHFLOW problem. In the standard scheduling classification scheme, the BATCHFLOW problem is written as $1|r_j, s_f = s, F = 1|\sum F_j$, where the part $s_f = s, F = 1$ refers to the fact that each job family has a fixed setup time and all jobs belong to the same family (see Potts and Kovalyov 2000). As special cases, we consider the problem variants with *identical processing times* $p_j = p$, and with a *fixed job sequence*, where jobs are to be processed in release order.

1.2 Online algorithms for a single machine with batching

In the online version of the BATCHFLOW problem, jobs are released over time, and any algorithm can base its batching decisions at a given time instant only on the jobs that have been released so far. We study the online problem under the *non-preemptive clairvoyant* setting: No information about a job is known until it is released, but once a job j has been released, both its release time r_j (that is, arrival time) and processing time p_j are known. A batch that has started processing cannot be stopped before completion.

In this paper we consider *deterministic* online algorithms. Without loss of generality, we assume that these algorithms have a particular structure, as described in the following.

From the problem definition it follows that no online algorithm can start a new batch as long as the machine is busy. Furthermore, any online algorithm needs to revise a decision only when new information becomes available, that is, when a new job is released. Therefore, we consider online algorithms that only take a decision at the completion time of a batch σ_u, or when a new job j is released and the machine is idle. We refer to these two events as triggering events. In either case, the algorithm bases its decision on the jobs $\{1, \ldots, j\}$ that have been released so far, and on the batches $\sigma_1, \ldots, \sigma_u$ created so far. Note that the set $\mathcal{P}$ of currently pending jobs can be deduced from this information.

In case of a triggering event, an online algorithm $\mathcal{A}$ takes the following two decisions. First, it tentatively chooses the next batch $\sigma_{\mathcal{A}}$ to be executed on the machine.

However, it does not execute the batch $\sigma_{\mathcal{A}}$ immediately. Rather, the algorithm's second decision defines a *delay time* $\Delta_{\mathcal{A}}$ by which it delays the execution of $\sigma_{\mathcal{A}}$, and waits for a triggering event to occur in the meantime. If $\Delta_{\mathcal{A}}$ time has elapsed, and no triggering event has happened, then the algorithm starts the batch $\sigma_{\mathcal{A}}$ on the machine (by definition, the machine is idle in this case). If, however, a triggering event occurs during the delay time, then the algorithm newly chooses $\sigma_{\mathcal{A}}$ and $\Delta_{\mathcal{A}}$. Thus, an online algorithm $\mathcal{A}$ is completely specified by how it chooses $\sigma_{\mathcal{A}}$ and $\Delta_{\mathcal{A}}$.

To evaluate different online algorithms for the online BATCHFLOW problem, we use competitive analysis: For a given problem instance I, let $\mathcal{F}_{\text{OPT}}(I)$ be the total flow time of an optimal solution, and $\mathcal{F}_{\mathcal{A}}(I)$ the total flow time of the solution obtained from some online algorithm $\mathcal{A}$. We are interested in the *strict competitive ratio* of online algorithm $\mathcal{A}$, defined as $\sup_I \frac{\mathcal{F}_{\mathcal{A}}(I)}{\mathcal{F}_{\text{OPT}}(I)}$. The online algorithm $\mathcal{A}$ is *c-competitive* if there is a constant α such that for all instances I, $\mathcal{F}_{\mathcal{A}}(I) \leq c \cdot \mathcal{F}_{\text{OPT}}(I) + \alpha$. When this condition holds also for $\alpha = 0$, we say that $\mathcal{A}$ is *strictly c-competitive*.

1.3 Related work

Since $\sum_{j \in J} r_j$ is a constant that we cannot influence, the offline version of our problem is equivalent to the problem $1|r_j, s_f = s, F = 1|\sum C_j$. The related problem $1|s_f = s, F = 1|\sum C_j$, without release times but with individual processing times, was first considered by Coffman et al. (1990). They solve this problem in $O(n \log n)$ time, first sorting the jobs by processing time, and then using an $O(n)$ dynamic programming algorithm. Albers and Brucker (1993) extend that solution to solve $1|s_f = s, F = 1|\sum w_j C_j$ for a fixed job sequence in $O(n)$ time, and show that the unrestricted problem $1|s_f = s, F = 1|\sum w_j C_j$ is unary NP-hard. Webster and Baker (1995) describe a dynamic program with running time $O(n^3)$ for the so-called *batch processing model*, where each batch has the same size-independent processing time, but the batch size is limited.

More recently, Cheng and Kovalyov (2001) describe complexity results for various related problems and objectives, also considering due dates, but not release times. They consider the *bounded* model, where the size of a batch[1] can be at most B, as well as the *unbounded* model.

The objective of minimizing the total completion time (weighted or unweighted) has been considered also in the so-called *burn-in* model (see Lee et al. 1992), where the processing time of a batch is equal to the maximum processing time among all jobs in the batch. For this model, Poon and Yu (2004) present two algorithms for $1|s_f = s, F = 1, B|\sum C_j$ with batch size bound B, with running times $O(n^{6B})$ and $n^{O(\sqrt{n})}$. Furthermore, Deng et al. (2004) consider $1|s_f = s, F = 1|\sum w_j C_j$ in the burn-in model with unbounded batch size. They show NP-hardness of that problem, and give a polynomial time approximation scheme.

Concerning the online setting, most previous work focuses on the burn-in model. An exception is Divakaran and Saks (2001), who consider the problem $1|r_j, s_f|\max F_j$ with sequence-independent setup times and several job families under job availability (i.e. the processing of each job completes as soon as its processing

[1] This bound is also called *capacity* by some authors.

time has elapsed). They present an $O(1)$-competitive online algorithm for that problem. For the burn-in model, Chen et al. (2004) consider the problem $1|r_j|\sum w_j C_j$, and present a 10/3-competitive online algorithm for unbounded batch size, as well as a $4+\varepsilon$-competitive online algorithm for bounded batch size.

The online problem $1|r_j|C_{\max}$ of minimizing the makespan in the burn-in model has been considered in several studies. Independently, Deng et al. (2003) and Zhang et al. (2001) gave a $(\sqrt{5}+1)/2$ lower bound for the competitive ratio, and both gave the same online algorithm for the unbounded batch size model which matches the lower bound. Poon and Yu (2005a) present a different online algorithm with the same competitive ratio, and describe a parameterized online algorithm which contains their own and the previous solution as special cases. Poon and Yu (2005b) give a class of 2-competitive online algorithms for bounded batch size, and a 7/4-competitive algorithm for batch size limit $B=2$.

Bein et al. (2004) propose optimally competitive online algorithms for the *list batching problem*. As in the BATCHFLOW problem, the goal is to minimize the total flow time of jobs. However there are no release times, or equivalently, all jobs are released at time zero. Still, the algorithm learns the jobs one after the other, and each time has to decide whether to include this job as the last in the current batch and start processing the batch, or to keep the batch open for later jobs.

1.4 Contribution and outline of the paper

To the best of our knowledge, we are the first to consider release times with the objective of minimizing the total flow time $\sum F_j$ under batch availability. We study this problem in an online setting, call it online BATCHFLOW problem, and introduce the GREEDY online algorithm in Sect. 2.1. For the special case of identical processing times p, we show that GREEDY is strictly 2-competitive, using the fact that its makespan is optimal up to an additive constant. In Sect. 2.3, we present two lower bounds, $1+\frac{1}{1+\frac{\max(p,s)}{\min(p,s)}}$ and $1+\frac{1}{1+2\frac{p}{s}}$ for this problem variant, and hence show that GREEDY is not far from optimal for this variant.

For the general online BATCHFLOW problem, we then give an $\frac{n}{2}-\varepsilon$ lower bound for the competitive ratio, and show that any online algorithm which avoids unnecessary idle time, including GREEDY, is strictly n-competitive, which matches the order of the lower bound.

2 The online BATCHFLOW problem

We first analyze the online BATCHFLOW problem for jobs with identical processing times $p_j=p$. For this case, we present a 2-competitive greedy algorithm in Sect. 2.1, and derive two lower bounds in terms of p and s for any online algorithm in Sect. 2.3. Next, Sect. 2.4 discusses bounds for any online algorithm for the case of general processing times.

2.1 The GREEDY batching algorithm for identical processing times

In this section, we consider the restricted case where all jobs have identical processing times $p_j = p$. This case is relevant in applications such as ours, where records of fixed length are to be logged. Note that the reordering of jobs with identical processing times is never beneficial, so it is irrelevant whether the fixed job sequence restriction is present or not, and we assume in the following that jobs are never reordered.

We now define the GREEDY algorithm, which always starts a batch consisting of all currently pending jobs as soon as the machine becomes idle.

Algorithm 1 GREEDY

Whenever the machine becomes idle:
  Set $\Delta_{\mathcal{A}} = 0$ (start the next batch immediately)
  Choose $\sigma_{\mathcal{A}} =$ the set of currently pending jobs

First, we focus on the makespan of GREEDY. Let us first understand how the structure of the GREEDY solution relates to alternative solutions that use fewer batches. To that end, we consider GREEDY's solution to a given instance and compare it to some other solution for the same instance, denoted by ANY. By definition, each of these two solutions consists of a sequence of consecutive batches. We require the following lemma, which is illustrated in Fig. 1.

Lemma 1 *Consider a sequence $\sigma_a, \ldots, \sigma_b$ of u* GREEDY *batches, and a sequence $\sigma'_c, \ldots, \sigma'_d$ of v* ANY *batches, such that the* ANY *sequence contains all the jobs in the* GREEDY *sequence (in the same order), and possibly additional jobs. If $u \geq v + 1$, then there exists a batch σ'_* among* ANY*'s batches that contains both at least one entire* GREEDY *batch, and at least one following job j_* from the next* GREEDY *batch.*

Proof Instead of proving the lemma directly, we prove the following equivalent statement: *Assuming that no batch among $\sigma'_c, \ldots, \sigma'_d$ fully contains a* GREEDY *batch plus a following job, it holds that $u \leq v$.*

We show that claim by induction over u. For $u = 1$ the claim is trivially true. Suppose that for $u \geq 2$, the claim holds for $1, \ldots, u-1$. By our assumption, the first ANY batch σ'_c can at most contain σ_a entirely, but no following jobs. Thus, the jobs in $\sigma_{a+1}, \ldots, \sigma_b$ must be covered by $\sigma'_{c+1}, \ldots, \sigma'_d$. These two sequences have lengths $u-1$ and $v-1$, respectively, so by the induction hypothesis, we have $1 + u - 1 \leq 1 + v - 1$, thus $u \leq v$. □

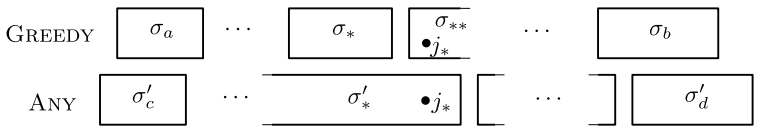

Fig. 1 Illustration of Lemma 1 and Lemma 2

The following lemma shows that if GREEDY needs time t to finish a set of batches $\{\sigma_1, \ldots, \sigma_u\}$, then no other algorithm can complete the same jobs before time $t - s$. Thus, GREEDY is 1-competitive for minimizing the makespan, with an additive constant $\alpha = s$.

Lemma 2 *For a given problem instance of the online* BATCHFLOW *problem with identical processing times, let $\sigma = \sigma_1, \ldots, \sigma_k$ with batch starting times $T_1, \ldots, T_k$ be the* GREEDY *solution, and let $\sigma' = \sigma'_1, \ldots, \sigma'_m$ with $T'_1, \ldots, T'_m$ be some other solution* ANY *for the same instance. For any batch $\sigma_u \in \sigma$ completing at time t, it holds that any batch $\sigma'_v \in \sigma'$ satisfying the condition*

$$\sum_{i=1}^{v} |\sigma'_i| \geq \sum_{i=1}^{u} |\sigma_i| \tag{1}$$

completes at time $t' \geq t - s$.

Proof For a given batch $\sigma_u \in \sigma$ consider the first batch $\sigma'_v \in \sigma'$ for which (1) holds. Such a batch exists because $\sum_{i=1}^{m} |\sigma'_i| = n$ and of course $\sum_{i=1}^{u} |\sigma_i| \leq n$ (note that $\sum_{i=1}^{u} |\sigma_i| = n$ holds only if σ_u is the last GREEDY batch, i.e., $u = k$). We assume that the GREEDY batches $\sigma_1, \ldots, \sigma_u$ are executed without any idle time in between. Indeed, if such an idle time occurs, GREEDY must have processed all jobs which have been released so far, and the idle time ends exactly at the release time of the next job. Of course, ANY cannot start processing this job earlier than GREEDY does. Hence, ignoring all jobs before such an idle time can only affect the comparison in favor of ANY.

First, we consider the case $u \geq v + 1$, where GREEDY uses at least one batch more than ANY. Apply Lemma 1 to $\sigma_1, \ldots, \sigma_u$ and $\sigma'_1, \ldots, \sigma'_v$, and let σ'_* be the *last* ANY batch containing a full GREEDY batch followed by at least one job. Choose σ_* such that it is the last full GREEDY batch whose jobs are contained in σ'_* that is followed by some job j_* in σ'_*. When j_* is released, GREEDY has already started processing batch σ_* (or has even finished), because otherwise j_* would be part of σ_*. On the other hand, ANY cannot start processing batch σ'_* before j_* is released. So, it must hold that $T_{\sigma_*} \leq T_{\sigma'_*}$. Let σ_{**} be the GREEDY batch following σ_*. Note that Lemma 1 (in contraposition) can be applied also to the sequences $\sigma_{**}, \ldots, \sigma_u$ and $\sigma'_*, \ldots, \sigma'_v$. Thus, since in these two sequences no ANY batch contains an entire GREEDY batch followed by another job, we have $|\{\sigma_{**}, \ldots, \sigma_u\}| \leq |\{\sigma'_*, \ldots, \sigma'_v\}|$. Defining z as the number of batches in $\{\sigma_*, \ldots, \sigma_u\}$, and z' as the number of batches in $\{\sigma'_*, \ldots, \sigma'_v\}$, it holds $z \leq z' + 1$. Putting all of the above together, we obtain:

$$t - t' \leq T_{\sigma_*} + p \cdot (|\sigma_*| + \cdots + |\sigma_u|) + zs - T_{\sigma'_*} - p \cdot (|\sigma'_*| + \cdots + |\sigma'_v|) - z's \leq s.$$

Finally, we consider the remaining case $u \leq v$. Since GREEDY has no idle time, it starts the first batch at T_1, which is when the first job is released. Note that ANY cannot start a batch before this time, hence $T'_1 \geq T_1$. GREEDY completes σ_u exactly at time $t = T_1 + p \cdot \sum_{i=1}^{u} |\sigma_i| + us$. ANY finishes σ'_v at time $t' \geq T'_1 + p \cdot \sum_{i=1}^{v} |\sigma'_i| + vs$ or later. So, using condition (1), we obtain that $t' - t \geq (v - u)s$, proving the theorem for any $u \leq v + 1$, and for $u \leq v$ in particular. □

Corollary 3 *The* GREEDY *online algorithm always computes a solution with makespan at most s larger than the minimum makespan.*

From this corollary, we obtain the following lemma.

Lemma 4 *In the online* BATCHFLOW *problem with identical processing times, consider any batch σ_u of the* GREEDY *solution, with starting time T_u. Let σ' be the first batch of the optimal solution* OPT *that contains some job in σ_u. The earliest time that* OPT *can finish processing the m jobs in $\sigma_u \cap \sigma'$ is $T_u + mp$.*

Proof Observe that, if we deleted all jobs in $\sigma_u \setminus \sigma'$ from the problem instance, then GREEDY would start processing exactly the m jobs in $\sigma_u \cap \sigma'$ in one batch at time T_u, and finish at $T_u + mp + s$. Now, if OPT were to finish these m jobs before $T_u + mp$, then there would exist a solution with makespan more than s smaller than GREEDY's makespan. This is a contradiction to Theorem 2. □

Note that the proofs of Theorem 2 and Lemma 4 can easily be adapted to incorporate non-identical processing times. We prove them for identical processing times here, since they serve as ingredients for the main theorem below, which only applies to identical processing times.

In order to compare an online solution against an optimal offline solution, let us now look at a special case of the latter.

Observation 5 *The total job flow time for optimally processing n jobs $1, \ldots, n$ with identical processing times $p_j = p$ and with identical release times is at least*

$$\mathcal{F}_n \geq \frac{1}{2}pn^2 + sn.$$

Proof Assume without loss of generality that all $r_j = 0$. Consider the first job: This job will have completion time at least $s + p$. The second job will finish no earlier than $s + 2p$, which can be achieved if the first two jobs are batched together. Generally, the i-th job can finish no earlier than $s + ip$, which would be achieved by batching the first i jobs together. This shows that $\mathcal{F}_n \geq \sum_{i=1}^{n}(s + ip) = \frac{1}{2}pn(n+1) + sn \geq \frac{1}{2}pn^2 + sn$. □

Theorem 6 *The* GREEDY *algorithm is strictly 2-competitive for the online* BATCHFLOW *problem with identical processing times.*

Proof Figure 2 shows all the relevant time instants for the proof. As in Lemma 4, consider any batch σ_u of size $l = |\sigma_u|$ of the GREEDY solution, with starting time T_u, and let σ' be the first batch of the optimal solution OPT that contains some jobs in σ_u. Further, let $m = |\sigma_u \cap \sigma'|$. Below, we compare the total accumulated flow time before and after time T_u for the jobs in σ_u, for both GREEDY and OPT.

Lemma 4 implies that no job in σ_u can complete before T_u in OPT. Thus, until time T_u, the jobs in σ_u have accumulated a total flow time of $\mathcal{F}^{\leq T_u}(\sigma_u) := \sum_{j \in \sigma_u}(T_u - r_j)$

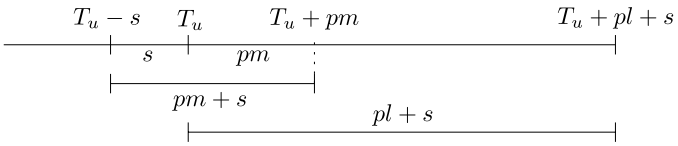

Fig. 2 Important time instants in GREEDY's competitiveness proof

in both GREEDY and OPT. Let $\mathcal{F}^{\geq T_u}_{\text{GREEDY}}(\sigma_u)$ denote the total flow time for the jobs in σ_u after time T_u in the GREEDY solution, and $\mathcal{F}^{\geq T_u}_{\text{OPT}}(\sigma_u)$ the same quantity for the OPT solution. Further, we let $\mathcal{F}_{\text{OPT}}(\sigma_u) = \mathcal{F}^{\geq T_u}_{\text{OPT}}(\sigma_u) + \mathcal{F}^{\leq T_u}(\sigma_u)$ and $\mathcal{F}_{\text{GREEDY}}(\sigma_u) = \mathcal{F}^{\geq T_u}_{\text{GREEDY}}(\sigma_u) + \mathcal{F}^{\leq T_u}(\sigma_u)$ be the total flow time for the jobs in σ_u in OPT and GREEDY, respectively. As σ' finishes at least pm time units after T_u (Lemma 4), and all jobs in σ_u must have been released by T_u, the total flow time of OPT for the jobs in σ_u after time T_u is

$$\begin{aligned}\mathcal{F}^{\geq T_u}_{\text{OPT}}(\sigma_u) &\geq \overbrace{m(pm)}^{\text{Lemma 4}} + \overbrace{(l-m)(pm)}^{\text{wait for } \sigma'} + \overbrace{\frac{1}{2}p(l-m)^2 + (l-m)s}^{\text{Observation 5}} \\ &= \frac{1}{2}(pl^2 + pm^2) + s(l-m). \end{aligned} \tag{2}$$

After time T_u, the GREEDY solution further accumulates a total flow time $\mathcal{F}^{\geq T_u}_{\text{GREEDY}}(\sigma_u) = P_u = l(lp+s)$ for the $l \geq m$ jobs in σ_u. Now, if σ' starts at $T_u - s$ or earlier, then all jobs in σ' must have been released at $T_u - s$ or earlier. Therefore, up until time T_u, the m jobs in $\sigma_u \cap \sigma'$ already yield an accumulated total flow time $\mathcal{F}^{\leq T_u}(\sigma_u) \geq sm$ in this case. Thus we have

$$\begin{aligned}\frac{\mathcal{F}_{\text{GREEDY}}(\sigma_u)}{\mathcal{F}_{\text{OPT}}(\sigma_u)} &\leq \frac{pl^2 + sl + \mathcal{F}^{\leq T_u}(\sigma_u)}{\frac{1}{2}(pl^2 + pm^2) + s(l-m) + \mathcal{F}^{\leq T_u}(\sigma_u)} \\ &= 1 + \frac{\frac{1}{2}p\overbrace{(l^2 - m^2)}^{\geq 0} + sm}{\frac{1}{2}p(l^2+m^2) + s\underbrace{(l-m)}_{\geq 0} + \underbrace{\mathcal{F}^{\leq T_u}(\sigma_u)}_{\geq sm}} \\ &\leq 1 + \frac{\frac{1}{2}p(l^2 - m^2) + sm}{\frac{1}{2}p(l^2+m^2) + sl}\end{aligned}$$

in this case.

Next, we consider the case in which σ' starts after $T_u - s$, say at starting time $T_u - s + \tau$, for $\tau > 0$. We still have $\mathcal{F}^{\geq T_u}_{\text{GREEDY}}(\sigma_u) = l(lp+s)$ for GREEDY. In this case, however, we obtain $\mathcal{F}^{\leq T_u}(\sigma_u) \geq m(s-\tau)$, and further an additive term $l\tau$ in

the bound (2) for $\mathcal{F}_{\text{OPT}}^{\geq T_u}(\sigma_u)$. Hence in this case,

$$\begin{aligned}
\frac{\mathcal{F}_{\text{GREEDY}}(\sigma_u)}{\mathcal{F}_{\text{OPT}}(\sigma_u)} &\leq \frac{pl^2 + sl + \mathcal{F}^{\leq T_u}(\sigma_u)}{\frac{1}{2}(pl^2 + pm^2) + s(l-m) + \tau l + \mathcal{F}^{\leq T_u}(\sigma_u)} \\
&= 1 + \frac{\frac{1}{2}p(l^2 - m^2) + sm - \tau l}{\frac{1}{2}p(l^2 + m^2) + s(l-m) + \tau l + \mathcal{F}^{\leq T_u}(\sigma_u)} \\
&\leq 1 + \frac{\frac{1}{2}p\overbrace{(l^2 - m^2)}^{\geq 0} + sm}{\frac{1}{2}p(l^2 + m^2) + s\underbrace{(l-m)}_{\geq 0} + \tau l + \underbrace{\mathcal{F}^{\leq T_u}(\sigma_u)}_{\geq m(s-\tau)}} \\
&\leq 1 + \frac{\frac{1}{2}p(l^2 - m^2) + sm}{\frac{1}{2}p(l^2 + m^2) + sl + \tau\underbrace{(l-m)}_{\geq 0}}.
\end{aligned}$$

So in both cases, we have

$$\frac{\mathcal{F}_{\text{GREEDY}}(\sigma_u)}{\mathcal{F}_{\text{OPT}}(\sigma_u)} \leq 1 + \frac{\frac{1}{2}p(l^2 - m^2) + sm}{\frac{1}{2}p(l^2 + m^2) + sl} = 1 + \frac{\frac{1}{2}(l^2 - m^2) + \frac{s}{p}m}{\frac{1}{2}(l^2 + m^2) + \frac{s}{p}l}.$$

Clearly, $\frac{\mathcal{F}_{\text{GREEDY}}(\sigma_u)}{\mathcal{F}_{\text{OPT}}(\sigma_u)} \leq 2$ because $m \leq l$. Furthermore, note that since l can be as large as n, this ratio comes arbitrarily close to 2 as n increases, irrespective of the values of s and p. Since $\frac{\mathcal{F}_{\text{GREEDY}}(\sigma_u)}{\mathcal{F}_{\text{OPT}}(\sigma_u)} \leq 2$ holds for any batch σ_u of the GREEDY solution, the theorem follows. □

2.2 A tight example for GREEDY for large n

Consider the following instance: The first job arrives at time 0, then immediately afterwards ($\varepsilon > 0$ later) a second job arrives. All other $k := n - 2$ jobs arrive at time $s + 2p + \varepsilon$. Greedy will process the first two jobs in two separate batches, and then all other jobs in one batch. Thus, the flow time of the GREEDY solution (omitting ε) is

$$\mathcal{F}_{\text{GREEDY}} = (s+p) + 2(s+p) + k(s+s+kp) = k^2 p + 2ks + 3s + 3p.$$

We compare this solution with the following: The first two jobs are processed together in one batch starting at ε. The other $k = n - 2$ batches are split into $\sqrt{k}$ batches of size $\sqrt{k}$ each (for simplicity, we assume that $k = n-2$ is a square number). The total flow time of this solution (again omitting ε) is

$$2(s+2p) + \sum_{i=1}^{\sqrt{k}} i\sqrt{k}(s + \sqrt{k}p) = 2(s+2p) + \frac{1}{2}k^2 p + \frac{1}{2}ks + \frac{1}{2}pk^{3/2} + \frac{1}{2}sk^{3/2}.$$

Hence we have

$$\frac{\mathcal{F}_{\text{GREEDY}}}{\mathcal{F}_{\text{OPT}}} \geq \frac{2k^2p + 4ks + 6(s+p)}{k^2p + ks + pk^{3/2} + sk^{3/2} + 4(s+2p)}$$
$$= \frac{2p + 4s/k + 6(s+p)/k^2}{p + s/k + p/k^{1/2} + s/k^{1/2} + 4(s+2p/k^2)},$$

and this ratio approaches 2 as n increases (recall $k = n - 2$). Thus, for arbitrary positive values of s and p, and for sufficiently large values of n, our analysis of the competitive ratio of GREEDY is tight.

2.3 Lower bounds for identical processing times

To complement the upper bound on the competitive ratio of our GREEDY algorithm, we derive two lower bounds on the competitive ratio of any algorithm for the online BATCHFLOW problem, again with identical processing times. These bounds show that no online algorithm can be much better than GREEDY for this setting.

For the following bounds, when we write that the adversary lets a job to be released *immediately after* t, we mean that the job's release time is $t + \varepsilon$ for an arbitrarily small $\varepsilon > 0$. For simplicity, we do not include ε in the calculations, but all proofs could be easily adapted by including ε and making it sufficiently small.

Theorem 7 *No online algorithm for the online* BATCHFLOW *problem with identical processing times can have a competitive ratio lower than*

$$1 + \frac{1}{1 + \frac{\max(p,s)}{\min(p,s)}} \leq \frac{3}{2}.$$

Proof Let $\mathcal{A}$ be any online algorithm with finite delay time $\Delta_{\mathcal{A}}$ for $\mathcal{P} = \{1\}$. The adversary chooses release times $r_1 = 0$, r_2 immediately after $\Delta_{\mathcal{A}}$, and r_j immediately after $\Delta_{\mathcal{A}} + p(j-1) + s(j-2)$ for $j \in \{3, \ldots, n\}$, as depicted in Fig. 3. Observe that an offline solution can avoid any waiting time for $n - 2$ jobs: If job 1 and job 2 are processed together, the first batch has finished just when job 3 is released, so if job 3 is processed immediately, it will be finished just when job 4 is released, and so on until job n. Thus,

$$\mathcal{F}_{\text{OPT}} \leq \overbrace{\Delta_{\mathcal{A}}}^{\text{job 1 waits}} + 2(2p+s) + (n-2)\cdot(p+s) = n(p+s) + \Delta_{\mathcal{A}} + 2p.$$

For bounding the flow time of $\mathcal{A}$'s solution, we examine for each job j the earliest possible completion time that $\mathcal{A}$ can achieve.

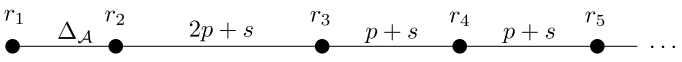

Fig. 3 The lower bound construction

By construction of the example, $\mathcal{A}$ cannot batch job 2 together with job 1, and starts processing job 1 at time $\Delta_{\mathcal{A}}$, which completes at $C_1 = \Delta_{\mathcal{A}} + p + s$. Hence, job 2 cannot start processing before C_1, and thus $C_2 \geq \Delta_{\mathcal{A}} + 2p + 2s$. By induction, we show that for each $j \in \{3, \ldots, n\}$, it holds $C_j \geq \Delta_{\mathcal{A}} + pj + s(j-1) + \min(p, s)$.
j = 3 : As batching job 3 together with later jobs will only increase job 3's completion time, the earliest possible completion time C_3 is either achieved by batching job 3 with job 2, or by processing job 3 separately. The two possibilities yield completion times

$$C_3' = \Delta_{\mathcal{A}} + 2p + s + \overbrace{2p + s}^{\text{process jobs 2, 3}} = \Delta_{\mathcal{A}} + 4p + 2s$$

and

$$C_3'' = \Delta_{\mathcal{A}} + 2p + 2s + p + s = \Delta_{\mathcal{A}} + 3p + 3s,$$

respectively. We have $C_3 \geq \Delta_{\mathcal{A}} + 3p + 2s + \min(p, s)$.
j − 1 → j : Note that batching more than two jobs would result in an idle time of more than $p + s$, which is certainly not fastest possible. So, the fastest possible way to process job j is to either batch it with job $j - 1$ or to process it separately. If job j is batched with job $j - 1$, then

$$C_j \geq r_j + 2p + s = \Delta_{\mathcal{A}} + p(j+1) + s(j-1).$$

If job j is processed separately,

$$\begin{aligned} C_j &\geq C_{j-1} + p + s \geq \Delta_{\mathcal{A}} + p(j-1) + s(j-2) + \min(p, s) + p + s \\ &= \Delta_{\mathcal{A}} + pj + s(j-1) + \min(p, s). \end{aligned}$$

Again, we see that $C_j \geq \Delta_{\mathcal{A}} + pj + s(j-1) + \min(p, s)$.

Adding $\sum_{j=1}^{n} F_j = \sum_{j=1}^{n} (C_j - r_j)$, we get

$$\begin{aligned} \mathcal{F}_{\mathcal{A}} &\geq \overbrace{\Delta_{\mathcal{A}} + p + s}^{\text{job 1}} + \overbrace{2p + 2s}^{\text{job 2}} + \sum_{j=3}^{n} (p + s + \min(p, s)) \\ &= \Delta_{\mathcal{A}} + np + p + ns + s + (n-2)\min(p, s). \end{aligned}$$

The competitive ratio can now be bounded as

$$\begin{aligned} \frac{\mathcal{F}_{\mathcal{A}}}{\mathcal{F}_{\text{OPT}}} &\geq \frac{\Delta_{\mathcal{A}} + np + p + ns + s + (n-2)\min(p, s)}{n(p+s) + \Delta_{\mathcal{A}} + 2p} \\ &\to \frac{p + s + \min(p, s)}{p + s} = 1 + \frac{1}{1 + \frac{\max(p,s)}{\min(p,s)}} \quad \text{for } n \to \infty. \end{aligned}$$

□

Using a similar construction, we get the following lower bound.

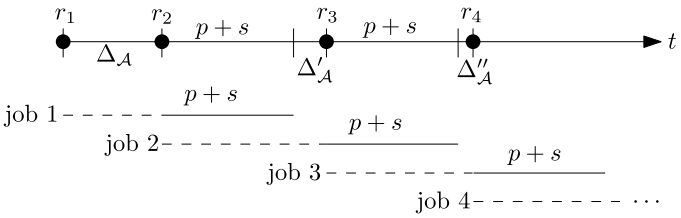

Fig. 4 Forcing $\mathcal{A}$ to process all jobs separately. Waiting time is shown as dashed lines, batch processing time as solid lines

Theorem 8 *No online algorithm for the online* BATCHFLOW *problem with identical processing times can have a competitive ratio lower than*

$$1 + \frac{1}{1 + 2\frac{p}{s}} \leq 2.$$

Proof The following adversary can force every online algorithm $\mathcal{A}$ to process all jobs separately: Release job 1 and wait for $\Delta_{\mathcal{A}}$ until $\mathcal{A}$ processes it, then immediately release job 2, wait for $\Delta'_{\mathcal{A}}$ until $\mathcal{A}$ processes it, and so on. Note that this causes every job except the first one to wait at least $p + s$ (see Fig. 4). Furthermore, the intervals between the job release times are all at least $p + s$ (and exactly $p + s$ if $\mathcal{A}$ processes each job as soon as the previous batch finishes, i.e. all $\Delta'_{\mathcal{A}} = \Delta''_{\mathcal{A}} = \ldots = 0$), except for the first interval $\Delta_{\mathcal{A}}$, which can be smaller. The flow time of $\mathcal{A}$'s solution thus equals the batch processing time plus at least $p+s$ waiting time for $n-1$ jobs, giving

$$\mathcal{F}_{\mathcal{A}} \geq \overbrace{n(p+s)}^{\text{processing}} + \overbrace{\Delta_{\mathcal{A}} + (n-1)(p+s)}^{\text{waiting}} = (2n-1)(p+s) + \Delta_{\mathcal{A}}.$$

For bounding the optimal offline flow time, we consider the solution of batching job 1 and job 2 together, and processing all other jobs separately. Since each interval after job 2 is at least $p+s$, and processing job 1 and job 2 together takes $2p+s$ time, job 3 needs to wait at most p. Processing job 3 will hence complete no later than $r_4 + p$, so job 4 has to wait at most p, and so on. So, we can bound the optimal offline flow time as

$$\begin{aligned}\mathcal{F}_{\text{OPT}} &\leq \Delta_{\mathcal{A}} + \overbrace{2(2p+s)}^{\text{process jobs } 1,2} + \overbrace{(n-2)p}^{\text{waiting of jobs } 3,\ldots,n} + \overbrace{(n-2)(p+s)}^{\text{process jobs } 3,\ldots,n} \\ &= \Delta_{\mathcal{A}} + 2np + ns.\end{aligned}$$

Thus, the competitive ratio is bounded by

$$\frac{\mathcal{F}_{\mathcal{A}}}{\mathcal{F}_{\text{OPT}}} \geq \frac{2p + 2s - \frac{p+s-\Delta_{\mathcal{A}}}{n}}{2p + s + \frac{\Delta_{\mathcal{A}}}{n}} \to 1 + \frac{1}{1 + \frac{2p}{s}} \quad \text{for } n \to \infty.$$

□

2.4 Bounds on the competitive ratio with job reordering

The following theorem shows that no online algorithm can have a good worst case performance for the online BATCHFLOW problem with general processing times, if the jobs do not need to be scheduled in the order they are given.

Theorem 9 *For the online* BATCHFLOW *problem, no online algorithm can have competitive ratio better than* $\frac{n}{2}$.

Proof Let $\mathcal{A}$ be any online algorithm. Consider an instance of n jobs, where job 1 has processing time p, and jobs $2, \ldots, n$ have processing time 1 each, and are released immediately after $\mathcal{A}$ starts processing job 1 (after having delayed for $\Delta_{\mathcal{A}}$ time units). Note that any competitive algorithm must start processing job 1 after some finite time (otherwise, its competitive ratio is unbounded if no further job arrives after job 1). As each of the jobs $2, \ldots, n$ has to wait for job 1 to finish, the total flow time for $\mathcal{A}$ is

$$\mathcal{F}_{\mathcal{A}} \geq \Delta_{\mathcal{A}} + n(p+s) + \frac{1}{2}(n-1)^2 + (n-1)s,$$

where we used the lower bound from Observation 5 for optimally processing $(n-1)$ jobs of equal processing time arriving at the same time.

For OPT, consider the solution that first processes jobs $2, \ldots, n$ in one batch, and after that processes job 1:

$$\mathcal{F}_{\text{OPT}} \leq \Delta_{\mathcal{A}} + n((n-1)+s) + p + s.$$

We assume in the following that $\Delta_{\mathcal{A}} \leq p+s$; if $\Delta_{\mathcal{A}} > p+s$, then our bound for $\mathcal{F}_{\mathcal{A}}$ increases, but we can decrease the bound for $\mathcal{F}_{\text{OPT}}$ because OPT can complete job 1 even before the other jobs are released, and then process all other jobs in one batch. We thus have

$$\mathcal{F}_{\mathcal{A}} \geq np + \frac{1}{2}n^2 + 2ns - n - s + \frac{1}{2} \quad \text{and} \quad \mathcal{F}_{\text{OPT}} \leq 2p + 2s + n^2 - n + ns.$$

It is easily verified that

$$\left(\frac{n}{2} - \varepsilon\right) \cdot \mathcal{F}_{\text{OPT}} \leq \mathcal{F}_{\mathcal{A}} \quad \text{if we choose } p \geq \frac{1}{4\varepsilon}(n^3 + n^2 s + 2n + 2s + 2\varepsilon n). \qquad \square$$

Theorem 9 shows that for the general setting, there is no online algorithm with a sub-linear competitive ratio. However, we will see in the following that all so-called non-waiting algorithms, a class to which the GREEDY algorithm belongs, are strictly n-competitive, i.e., are at most a factor 2 away from the lower bound. We call an online algorithm *non-waiting* if it never produces a solution in which there is idle time while some jobs are pending.

Theorem 10 *Any non-waiting online algorithm for the online* BATCHFLOW *problem is strictly n-competitive.*

Proof Let $\mathcal{A}$ be any non-waiting online algorithm. Consider any job i, and let σ_u be the batch which contains job i. Furthermore, let J' be the set of all jobs not contained in σ_u. The flow time of job i is $F_i = C_i - r_i = T_u + P_u - r_i$. The longest possible interval during which σ_u needs to wait (i.e. the machine is busy) in a non-waiting algorithm's solution is $s|J'| + \sum_{j\in J'} p_j$. So, for a non-waiting algorithm, $T_u - r_i \leq s(n-1) + \sum_{j\in J'} p_j$. Hence,

$$F_i \leq P_u + s(n-1) + \sum_{j\in J'} p_j \leq sn + \sum_{j=1}^{n} p_j.$$

Thus, the total flow time for $\mathcal{A}$'s solution is

$$\mathcal{F}_{\mathcal{A}} = \sum_{i=1}^{n} F_i \leq n^2 s + n \cdot \sum_{j=1}^{n} p_j.$$

We now turn to the optimal solution OPT. Clearly, each job j has flow time $F'_j \geq p_j + s$, as the batch processing time is inevitable. Thus, the flow time of OPT is at least

$$\mathcal{F}_{\text{OPT}} = \sum_{j=1}^{n} F'_j \geq ns + \sum_{j=1}^{n} p_j.$$

Comparing the total flow times $\mathcal{F}_{\mathcal{A}}$ and $\mathcal{F}_{\text{OPT}}$ completes the proof. □

Observe that this upper bound proof does not make use of the fact that the reordering of jobs is allowed. Thus, adding a fixed job sequence constraint does not affect the validity of Theorem 10. Note that this is not true for Theorem 9.

We remark that the online non-preemptive scheduling problem with release times known from the literature (see e.g. Epstein and van Stee 2003) is a special case of the online BATCHFLOW problem. Thus, the $\Theta(n)$ upper bound for the former problem is implied by our Theorem 10.

3 Discussion

We studied the online BATCHFLOW problem $1|r_j, s_f = s, F = 1|\sum F_j$, and investigated this problem for the case of identical processing times and for general processing times.

We shortly mention here further results we can prove (see Gfeller et al. 2006), whose detailed explanations and proofs are omitted in this paper. With a fixed job sequence (but arbitrary processing times), no online algorithm for the online BATCHFLOW problem can have a constant competitive ratio. The general offline BATCHFLOW problem is NP-complete, even with machine setup time $s = 0$. When the job sequence is fixed, there exists a dynamic programming-like algorithm with polynomial running time.

References

Albers S, Brucker P (1993) The complexity of one-machine batching problems. Discrete Appl Math 47:87–107

Bein W, Epstein L, Larmore L, Noga J (2004) Optimally competitive list batching. In: 9th Scandinavian workshop on algorithms theory (SWAT). Lecture notes in computer science, vol 3111. Springer, Berlin, pp 77–89

Chen B, Deng X, Zang W (2004) On-line scheduling a batch processing system to minimize total weighted job completion time. J Comb Optim 8:85–95

Cheng T, Kovalyov M (2001) Single machine batch scheduling with sequential job processing. IIE Trans Sched Logist 33:413–420

Coffman E, Yannakakis M, Magazine M, Santos C (1990) Batch sizing and job sequencing on a single machine. Ann Oper Res 26:135–147

Deng X, Poon C, Zhang Y (2003) Approximation algorithms in batch processing. J Comb Optim 7:247–257

Deng X, Feng H, Zhang P, Zhang Y, Zhu H (2004) Minimizing mean completion time in a batch processing system. Algorithmica 38(4):513–528

Divakaran S, Saks M (2001) Online scheduling with release times and set-ups. Technical Report 2001-50, DIMACS

Epstein L, van Stee R (2003) Lower bounds for on-line single-machine scheduling. Theor Comput Sci 299(1–3):439–450

Gfeller B, Peeters L, Weber B, Widmayer P (2006) Single machine batch scheduling with release times. Technical Report 514, ETH Zurich, April 2006

Lee C, Uzsoy R, Martin-Vega L (1992) Efficient algorithms for scheduling semiconductor burn-in operations. Oper Res 40(4):764–775

Poon C, Yu W (2004) On minimizing total completion time in batch machine scheduling. Int J Found Comput Sci 15(4):593–607

Poon C, Yu W (2005a) A flexible on-line scheduling algorithm for batch machine with infinite capacity. Ann Oper Res 133:175–181

Poon C, Yu W (2005b) On-line scheduling algorithms for a batch machine with finite capacity. J Comb Optim 9:167–186

Potts C, Kovalyov M (2000) Scheduling with batching: a review. Eur J Oper Res 120:228–249

Webster S, Baker K (1995) Scheduling groups of jobs on a single machine. Oper Res 43(4):692–703

Zhang G, Cai X, Wong C (2001) On-line algorithms for minimizing makespan on batch processing machines. Nav Res Logist 48:241–258